Medium-speed multipliers trim cost, shrink bandwidth in speech transmission

Dedicated multiplier ICs can help you steer clear of the high cost and wide bandwidth requirements of digitally transmitting human voice signals. What's more, you won't even need the fastest multiplier chips—medium-speed devices give surprisingly high speed at much lower cost than their high-speed, parallel-type counterparts.

Digital voice transmission is usually accomplished with pulse-code modulation techniques (PCM), by sampling an analog speech signal at an 8-kHz rate with 8-bit resolution. But that requires a very broad bandwidth—64 kbits/s. There is an alternative. With a speech compression system, you can knock bandwidth down to just 2.4 kbits/s. But to design such a system, you first must understand how to interface a dedicated multiplier to a microprocessor.

Speech compression is one of the newest processes to take advantage of a medium-speed dedicated multiplier. However, anytime you have a design problem that doesn't call for the under 200-ns speeds of parallel multipliers, yet demands faster multiplication than you get from a traditional add-and-shift software algorithm, look to a medium-speed chip.

You should already have a good idea of the performance you can expect from your μP. Before you attempt to design a μP-multiplier interface for a speech-compression system, you should have an equally good idea of what a multiplier can do.

Medium speed doesn't mean slow

Medium-speed multipliers, such as an AMD 25LS2516 (Advanced Micro Devices) or a MMI 67508 (Monolithic Memories), multiply two 8-bit numbers in a mere 400 ns—worst case. As you might expect, a 16-bit multiplier like MMI's 67516 needs twice that time, 800 ns—also worst case. If you must multiply even faster, check out other devices and the special techniques explained in "Dedicated Multiplier ICs Speed-Up Processing In Fast Computer Systems," (ED No. 19, September 13, 1978, pp. 98-103).

Shlomo Waser, Product Planning Manager, Monolithic Memories, Inc., 1165 E. Arques Ave., Sunnyvale, CA 94086 and **Dr. Allen Peterson,** Professor of Electrical Engineering, Stanford University, Stanford, CA 94305.

Most medium-speed multipliers operate in sequential fashion, which means they generate partial products step-by-step. By contrast, the faster parallel devices generate products in a single step. But high-speed multipliers, which receive operand inputs in parallel, are expensive and large (they're built in 40-pin packages). Medium-speed devices are housed in 24-pin packages, and their cost is much easier to justify in designs that call for a dedicated multiplier. Another point to keep in mind is that you'll get double-duty out of many multiplier chips because they also perform division.

In a 67516, two operands are loaded into registers in a time sequence. The device then jumps to either the multiply or divide routine to carry out the arithmetic process its instruction calls for.

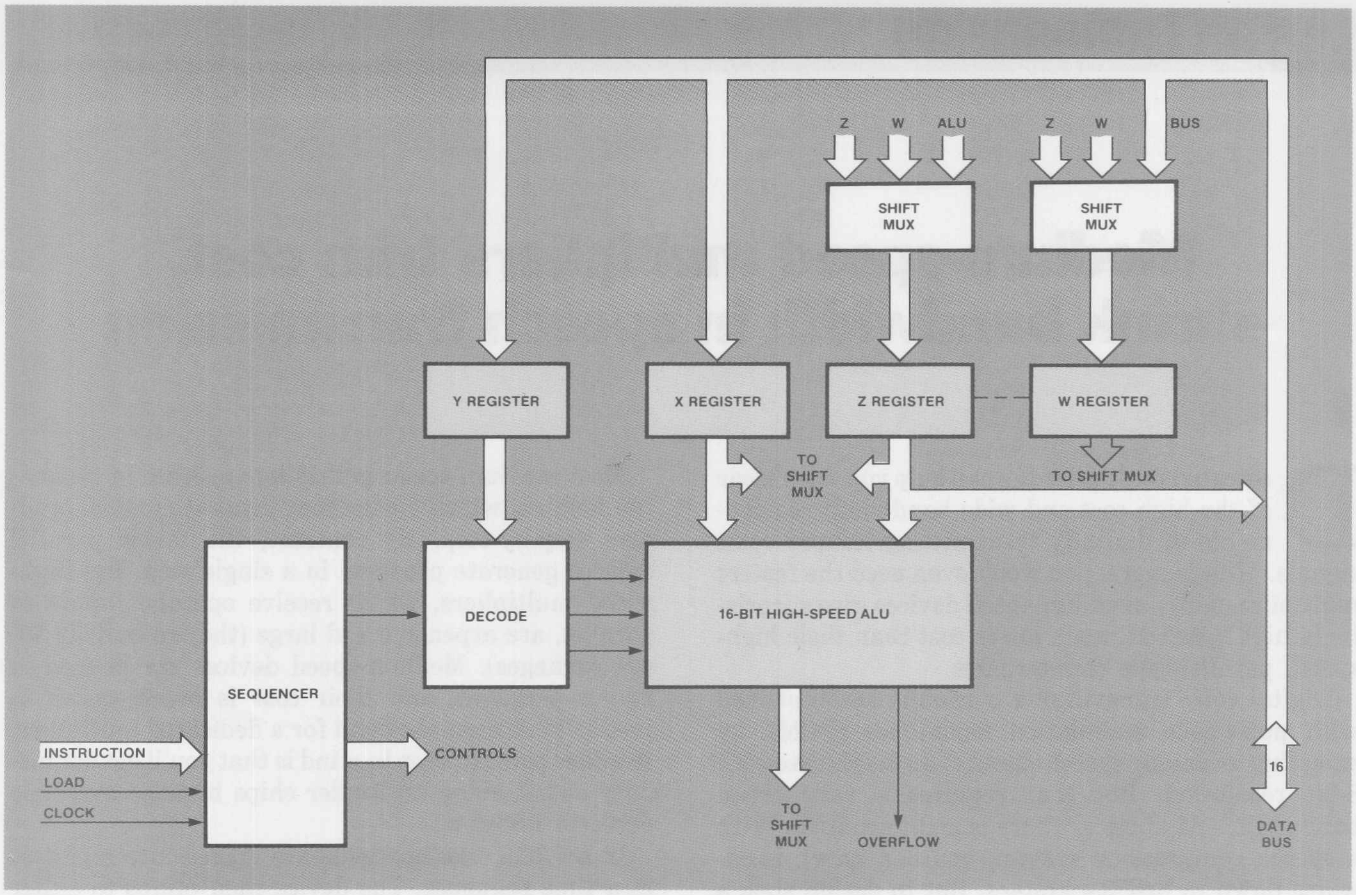
To understand the basic operation of this chip, look at the block diagram in Fig. 1. The device contains four working registers: The Y (multiplier), X (multiplicand/divisor), W (least-significant half of the double-length accumulator) and Z (most-significant half of the double-length accumulator). The last two registers are usually grouped together as the W-Z register, and operate as a working register for intermediate results. In addition, the W-Z register stores the final double-length product in multiplication or the quotient in division.

Final products or quotients are placed on the output bus in a time sequence, after the expiration of the number of clock cycles required to complete the operation. Multiplication requires eight clock-cycles, while division needs 20—both measured from the time operands are first loaded into the X and Y registers.

Three instruction lines, I_0, I_1 and I_2 select the function that the 67516 performs. You have a choice of sixteen multiply and seven divide options. For your option to be exercised, instructions must come from the microprocessor, so you'll have to know how μP instructions command a multiplier.

Make it easy on the μP

One of the primary benefits gained from using a dedicated multiplier is a savings of processor time. Your system will be capable of multiplying with a minimum of μP instructions since a dedicated multi-



1. **Four working registers and a 16-bit high-speed ALU** are the heart of Monolithic Memories' 67516 16-bit multiplier IC. The sequencer coordinates the three-line instruction code, which determines how the ALU operates on data.

plier needs far fewer instructions than if multiplication were to be performed in the processor's arithmetic logic unit (ALU).

Here's how a 67516 multiplier operates under processor command to perform the familiar sum-of-products operation represented by the expression,

$$\sum_{i=1}^{n} X_i Y_i$$

You'll need instruction codes sent from the μP to the multiplier (see Fig. 2 for a general scheme). An instruction code of 5 or 6 results in loading the first operand into the X register, depending on whether the operand is an integer or fractional. An instruction code of O follows, which tells the 67516 to load the next operand on the bus into the Y register. For the next eight clock cycles, the two operands are multiplied together, with the product entering the W-Z register. Note that your microprocessor can attend to other business during this time.

At the conclusion of the first multiplication, (end of the tenth clock cycle), two more instructions must be issued to the multiplier. An instruction code of 6 loads the next operand into the X register, and an instruction code of 2 tells the device to multiply and add the result to the contents of the W-Z register. Again, eight clock cycles must elapse before this result appears in the W-Z register.

Naturally, you can continue to issue 6 and 2 instruc-

OPERATION		TIME SLOT (cycle) 1	2	3	4 22	23
Z. W/X	INS CODE	6	6	4	DIVIDE	
	BUS	X	Z	W		

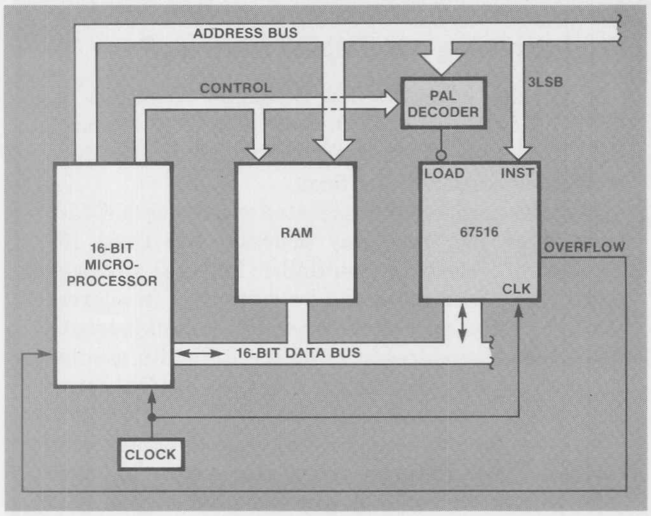
2. **Multiplication codes and computation times** show a sum-of-products operation in a 67516 chip. The second operation is the same as the first, only with a negative multiplication operation.

OPERATION		TIME SLOT (CYCLE) 1	2	3 10	11	12	13.........20
$XY + K_Z, K_W$	INS CODE	6	0	MULTIPLY	6	2	MULTIPLY
	BUS	X	Y		X_{i+1}	Y_{i+1}	
$-XY + K_Z, K_W$	INS CODE	6	1	MULTIPLY	6	3	MULTIPLY
	BUS	X	Y		X_{i+1}	Y_{i+1}	

3. **Codes and computation times** for a 67516's division operation require more time than multiplication. Division takes twenty time-cycles, multiplication takes eight. Division also requires an additional instruction code.

tions until the entire multiplication-summing process is complete. Then the result in the accumulator gets placed on the data bus.

Division and multiplication are similar—division just takes longer. The instructions and timing requirements for dividing a double-length dividend by a single-length divisor are shown in Fig. 3. For division, the processor must issue three instructions instead of



4. **Interfacing a dedicated multiplier to a 16-bit microprocessor** is a simple process. The least-significant three bits of the CPU instruction word forms the multiplier's instruction code. The remaining address bits determine whether the multiplier is selected.

two, and the divide cycle takes 20 time-slots rather than eight. One code is common to both divide and multiply operations—a 7 instruction-code reads the contents of the W-Z register to the data bus.

That takes care of the basic instructions needed to operate a multiplier. The next phase is interfacing—how the instructions get from μP to multiplier.

Interfacing the multiplier

A 67516 has only three instruction lines, so it's quite simple to connect to any 16-bit microprocessor (see Fig. 4). The technique is to assign the multiplier's three instruction lines to the three least-significant bits on the address bus, and route the bits to the instruction input. The remaining bits of the address bus are routed through a programmable array logic (PAL) device that acts as a decoder. Decoding these bits determines whether the 67516 is selected.

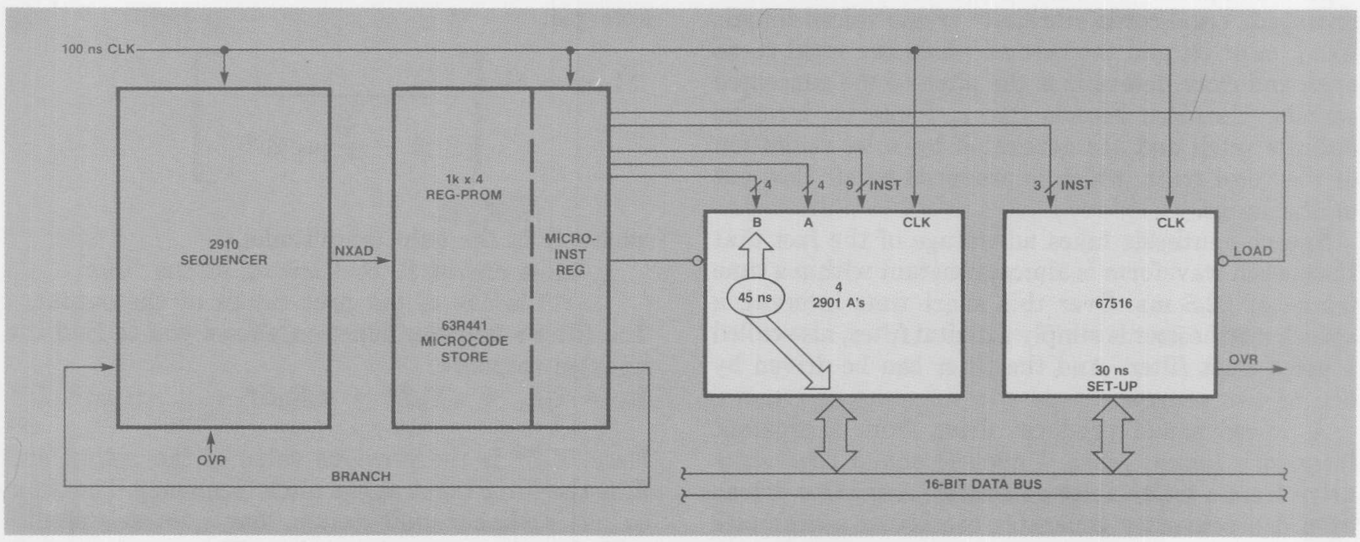
With this interface, if the multiplier is assigned to address location 100, any address in the 100 to 107 range selects the multiplier by enabling the chip. What actually happens is that the LOAD line goes low. The three least-significant bits then represent the instruction to be carried out. For example, if the CPU sends out address 106, the multiplier carries out the 6-instruction code, which is LOAD. Similarly, if instruction code 100 is sent out, the multiplier recognizes the 0-instruction, or multiply. If you want further details of this technique, called memory-mapped programming, refer to the box, "Programming A Multiplier From A μP."

At this point, you may be considering the tradeoff between programming a multiplier or allowing your system to multiply under the control of a processor's built-in multiply macroinstruction. Many of the new 16-bit, and some of the 8-bit μPs have this feature. Don't consider too long. If you're after speed, a 67516 can multiply twenty times faster than the newest 16-bit processor using a macroinstruction.

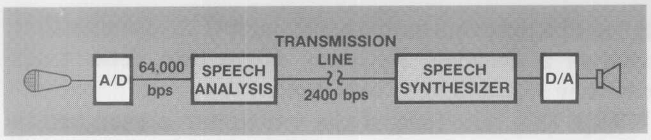
Multiplier interfacing can also be accomplished with a bit-slice microprocessor like AMD's 2901A (see Fig. 5). This system runs on a fast 100-ns clock, so you can use a two-stage pipeline technique, in which the first stage is an address register and the second is a PROM output register.

In bit-slice operation, a microinstruction located in the PROM tells the 2901A and the 67516 what operation to perform. Pipelining allows input data to be queued so the 67516's 800-ns multiply time can be worked in with the 100-ns system clock time. But this design only works well where relatively few branching decisions are made.

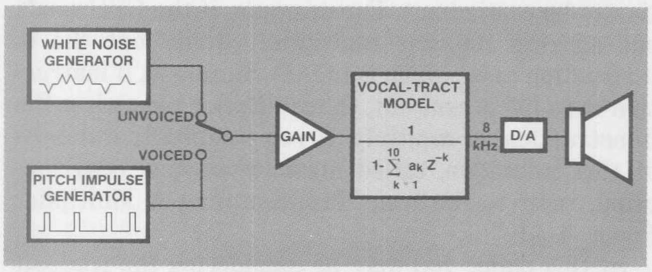
With the link between the microprocessor and multiplier established, the more difficult problem



5. **Another way of interfacing a multiplier** is with a bit-slice microprocessor. This pipeline configuration has both processor (a 4-bit slice) and multiplier interfaced directly to the 16-bit bus.



6. **A speech compression system** breaks down into two subsystems—analysis and synthesis.



7. **Speech synthesis,** shown in this digital model of a narrowband speech synthesizer, has three varying parameters—filter coefficients (a_K), gain and pitch period.

Programming a multiplier from a μP

With the μP-multiplier interface technique illustrated in Fig. 4, you can write a program that not only allows the μP to select the multiplier but tells it what operation to perform.

If you've assigned the dedicated multiplier (a 67516) to address 100, then any address code from 100 through 107 selects the multiplier. Let's say you have stored the two numbers to be multiplied in address locations 108 and 109, and you want the double-length result placed in addresses 110 and 111. Finally, assume that the μP MOVE instruction takes the general form,

MOV SA, DA,

where SA is the source address and DA is the destination address. The following segment of assembly language code will cause the multiplier to perform the MOVE operation:

Assembly Program	67516 Instruction
MOV 108,106	LOAD X (instruction 6)
MOV 109,100	LOAD Y (instruction 0)
NOP	MULTIPLY
MOV 107,110	Read 16 MSB of product (instruction 7)
MOV 107,111	Read 16 LSB of product (instruction 7)

When you write your program, you may not need the NOP instruction, depending on the speed of your processor. Without a NOP, some processors have more than a 800-ns delay between the WRITE pulse of the second instruction and the READ pulse of the fourth instruction. If your μP is this slow, omit the NOP. But if you use a very fast processor, you may need one or more NOPS to insure that the 67516's 800-ns throughput time is met.

comes to the forefront—compressing speech signals so they can be digitally transmitted.

Fundamentals of speech

Speech signals are produced by relatively slow movements of human vocal cords. This allows them to be described by parameters having a much lower information rate than the 64-kbit/s of PCM. Although speech compression circuits operate at a slow 2.4-kbits/s, you'll still have to do a lot of number crunching to compute the necessary parameters in real time. Once again, speed becomes essential—that means you'll need a dedicated multiplier to assist the microprocessor.

The algorithms for speech compression are fairly complex, so it's best to break the subject into two parts —speech synthesis and speech analysis (see Fig. 6). Synthesis is the easier of the two to understand.

Human speech is characterized as either voiced or unvoiced. Vocal cords vibrate to create voiced sounds like l, m or ee, and the rate at which the vocal cords open and close determines the pitch of the generated sounds. Unvoiced sounds like s, f and sh have no definite pitch and are generated by noise excitation of the vocal tract, which is produced by air flow out of the lungs.

Speech synthesis takes advantage of the fact that the speech waveform is almost constant within a time frame of 22.5 ms. Over this short time interval, a speech synthesizer is simply a digital filter, also called a voice tract filter. And the filter can be driven by one of two sources.

A voiced sound produces drive from a constant frequency (pitch) pulse. Unvoiced sound represents drive from a white noise generator. A speaker driven by a d/a converter generates the actual speech (see Fig. 6)—the converter is updated every 125 μs, an 8-kHz rate. Acceptable speech reproduction requires at least a 4-kHz bandwidth.

Speech synthesis is illustrated by the narrow-band speech synthesizer in Fig. 7. From this, a vocal tract filter model with the following transfer function emerges:

$$H(z) = G\left[\frac{1}{1-\sum_{k=1}^{10} a_k Z^{-k}}\right] \qquad (1)$$

$$= E_o/E_i$$

where G is the gain (amplitude),

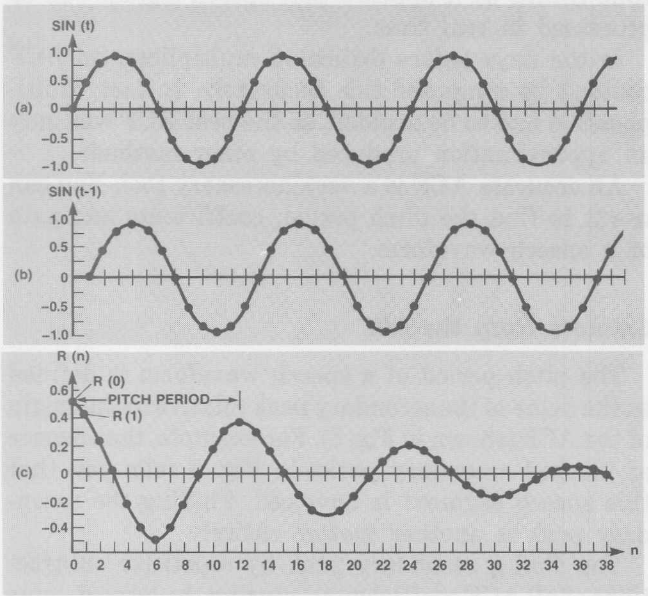
a_k is one of 10 coefficients of the filter,

z^{-k} is one of ten past values of the output.

The filter's transfer function allows you to find the impulse response,

$$E_O = GE_i + a_1E_OZ^{-1} + a_2E_OZ^{-2} + ... + a_{10}E_OZ^{-10}. \qquad (2)$$

Here, E_oZ^{-1} is the previous value of the output and E_i is the filter input at the pitch frequency. The filter output is a linear combination, that is, linearly predictable from its ten previous values. This speech processing scheme is called LPC-10, or Linear Predictive Code



8. **In an autocorrelation function (ACF),** the top sinusoid is multiplied by the middle one. The middle waveform is identical to the top, but delayed by one time unit. This product gives the R(0) coefficient shown in the bottom waveform. The ACF is composed of all coefficients, R(0) through R(n). The pitch period is the delay measured from the start of the ACF to the first peak.

using 10 coefficients.

The implication of Eq. 2 is that to synthesize speech, you only need eleven multiply and accumulate operations every 125 μs. Therefore, each multiply and accumulate can be as slow as 11.3 μs (125/11), and yield speech of acceptable quality. For this reason, speech synthesizers can be implemented with one of the older, slower semiconductor processes—PMOS (see the article "Here's a Breakthrough—A Low-Cost Speech Synthesizer On A Chip," ED No. 15, July 19, 1978, p. 32).

Although the speech waveform remains constant in a 22.5 ms time frame, within each time frame you must update the following parameters:

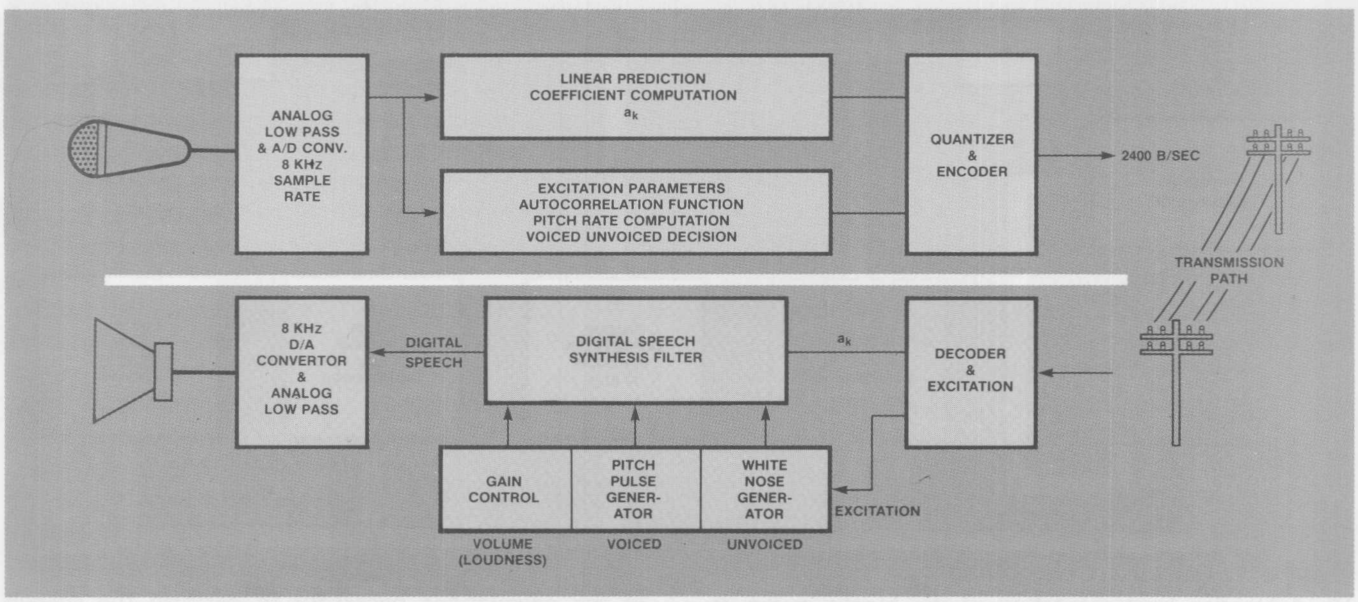
1. The pitch period (for a voiced sound).
2. The ten coefficients, a_1 through a_{10}.
3. The gain factor, G.

Some speech synthesizers store these parameters in a large ROM (256 kbits). At 2400 bits/s, that's equivalent to a speech duration of more than 100 s. But if you want to do narrowband speech transmission, you have to compute parameters on the fly, in real time. And that puts you into the province of speech analysis.

Speech analysis—delaying tactics

To determine the pitch period and the filter coefficients requires some higher mathematics—the autocorrelation function (ACF). The ACF coefficients are used to compute the pitch period and the ten coefficients. It turns out that the n^{th} coefficient of the ACF results from multiplying the speech waveform of one time frame by the same waveform delayed by n-samples (see Fig. 8).

At the top is a stored waveform (a sinusoid), and below it is the same sinusoid delayed by one time unit. The bottom waveform represents the collection of all ACF coefficients. Coefficient R(O) in the bottom waveform results from the top sinusoid being multiplied by itself. And R(l) results from the top signal being multiplied by the second waveform (1 unit delayed). Each new coefficient, R(n), is formed as the top waveform shifts n-units to the right. As this occurs, n-zeros are added on the left. The declining



9. **This speech compression system transmits voice** signals at 2400 bits/s. Designed with linear predictive code methods, the upper block is the analysis section, the lower is the synthesizer.

peaks of the ACF are caused by the shifting-in of zeros, from the left.

In this application, a single time frame contains 180 samples. You arrive at this number by dividing the 22.5 ms time frame by the 125 μs sampling period. In practice, it's usually not necessary to compute the entire ACF of a sample block (22.5 ms), since pitch periods normally fall in the range of 3 to 12 ms (pitch frequencies of 80 to 300 Hz). Specifically, it's sufficient to compute only the first 100 R's (called lags in speech jargon) of the ACF.

To understand this better, examine the ACF for 180 samples with 100 lags. For n = 0 to 99 or 100 lags,

$$R(n) = \frac{1}{180} \sum_{j=1}^{180} S_j S_{j-n}. \quad (3)$$

Here, S_j is the j^{th} sample of a speech waveform and R(n) is the n^{th} ACF coefficient. To see how long it takes to compute 100 lags, Eq. 3 must be expanded.

$$R(0) = 1/180\ (S_0S_0 + S_1S_1 + S_2S_2 + \ldots + S_{180}S_{180})$$
$$R(1) = 1/180\ (S_0O + S_1S_0 + S_2S_1 + \ldots + S_{180}S_{179})$$
$$\vdots$$
$$R(99) = 1/180\ (S_0O + S_1O + S_2O + \ldots + S_{180}S_{81}).$$

The first thing to notice about this expansion is that R(0) requires 180 multiply and accumulate operations, but R(99) needs only 80 such operations because of the shifted-in zeros. The average number of operations is 130 or (180 + 80)/2. Multiply the average number of operations by 100 lags and you get a total of 13,000 operations. If each multiply and accumulate can be accomplished in 1 μs, it will take only 13 ms to compute all 13,000 coefficients. That's well within the 22.5 ms time frame, so it appears that speech signals can be processed in real time.

In the days before dedicated multipliers, an ACF couldn't be computed this accurately. In fact, multiplication had to be avoided, so the best ACF was only an approximation produced by other methods.

An accurate ACF is a very necessary tool. You can use it to find the pitch period, coefficients and gain of a speech waveform.

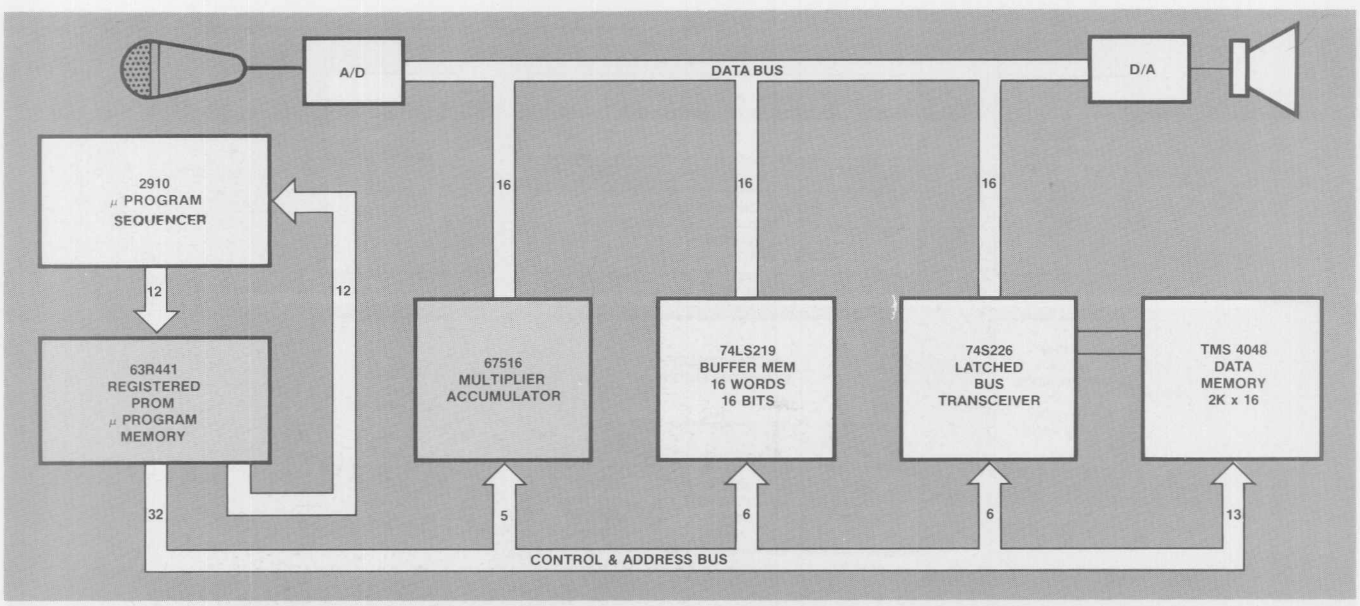
Spinoffs from the ACF

The pitch period of a speech waveform is defined as the delay of the secondary peak relative to the origin of the ACF (shown in Fig. 8). For example, the absence of marked secondary peaks in Fig. 8 tells you that this speech segment is unvoiced. Finding the secondary peak is another matter entirely.

You find a secondary peak by repetitive subtractions of all ACF coefficients—storing the largest value while scanning all 100 lags. These 100 subtractions are performed by a dedicated multiplier like the 67516. It multiplies each operand by 1, then subtracts the value from the accumulator. Each subtraction takes about 1 μs, which may seem slow, but all 100 subtractions take only 0.1 ms—insignificant within a 22.5 ms time frame. Even better, a dedicated multiplier performs the entire operation without any additional support from adders or the ALU.

The computation of filter coefficients isn't as simple as finding the pitch period. A set of linear equations specifies the ten filter coefficients.
For i = 1 to 10,

$$a_k = \sum_{R=1}^{10} a_K\, R(i - K) \quad (4)$$



10. **A bit-slice implementation** of the blocks in Fig. 9 requires the 67516 multiplier chip's full capability. Most of the calculations necessary to perform the speech synthesis function are done in the multiplier.

$a_k = R\ (i)$

Expanding this summation, you get,

$$
\begin{aligned}
a_1R(0) + a_2R(1) + a_3R(2) + \ldots + a_{10}R(9) &= R(1)\\
a_1R(1) + a_2R(0) + a_3R(1) + \ldots + a_{10}R(8) &= R(2)\\
&\ \ \vdots\\
a_1R(9) + a_2R(8) + a_3R(7) + \ldots + a_{10}R(0) &= R(10),
\end{aligned}
$$

where the a_k's are the 10 unknowns in this set of equations. It's possible to solve these equations in a few seconds with a general purpose computer. But who's got a few seconds? You have to solve this problem in a few milliseconds at most. The solution is an iterative method whose complete details can be found in the box "The Levinson/Durbin Recursion." With the aid of this algorithm, it's possible to compute all coefficients in the remarkable time of 0.35 ms.

Finally, you must compute the gain factor, G. This is a fairly simple problem that takes just 20 μs to perform with the following equation:

$$G^2 = R(0) - \sum_{R=1}^{10} a_k\ R(K).$$

At this point, only one thing really matters. With all the required computations, will the speech compression system operate in real time?

Total the time

The answer is found by adding together all the elements that make up the computation time for a single frame. If the total time used comes in under the maximum of 22.5 ms, the system will operate in real time.

For speech synthesis: 1.98 ms (180 outputs $\times$ 11 μs)
For speech analysis:

autocorrelation	13.0 ms
pitch period	0.1 "
filter coeff.	0.35 "
gain	0.02 "
Grand total:	15.45 ms

Only 15.45 ms of the 22.5-ms time frame has been used for all the computations. You've got a comfortable margin of 7 ms left over for serializing,

The Levinson/Durbin recursion

A major problem in speech analysis/synthesis is the determination of the values of ten filter coefficients in real time. These coefficients are related to the autocorrelation function's coefficients by the ten linear equations shown in the text as Eq. 4.

One of the advantages of using the linear predictive method to design speech compression systems is the availability of a very efficient iterative algorithm for finding the ten coefficients—the Levinson/Durbin recursion. To solve the ten equations in ten unknowns, start with the Levinson/Durbin definition,

$$k_i = \frac{\sum_{j=1}^{i-1} a_j(i-j)R(i-j) - R(i)}{E_{i-1}}$$

$$a_i(i) = -k_i$$

$$a_j(i) = a_j(i-1) + k_i a_{i-j}(i-1)$$

$$E_i = E_{i-1}(1-k_i^2).$$

This set of equations is solved recursively for i = 1, 2, . . . 10, with the final solution given by,

$$a_j = a_j(10) \text{ where } j = 1, 2, \ldots . 10.$$

The recursion is started by $E_0 = R(0)$, and from this you find k_1.

$$k_1 = -\frac{R(1)}{E_0} = -\frac{R(1)}{R(0)}$$

Then k_1 and E_0 are used to compute E_1 and $a_1(1)$.

$$a_1(1) = -k_1$$

$$E_1 = E_0(1 - k_1^2)$$

The iteration for k_2 is similar to that for k_1:

$$k_2 = \frac{a_1(1)R(1) - R(2)}{E_1}$$

$$E_2 = E_1(1 - k_2^2)$$

$$a_1(2) = a_1(1) + k_2 a_1(1).$$

Continue the iteration process until you get a value for k_{10}:

$$k_{10} = \frac{a_1(9)R(9) + a_2(9)R(8) + \ldots + a_9(9)R(1) - R(10)}{E_9}$$

Note that k_{10} is computed by ten multiply-and-accumulate operations and one division. To compute E_{10}, you'll have to perform three multiplications and one addition operation. And $a_1(1)$ through $a_9(9)$ take nine multiplications and nine additions. It turns out that the tenth iteration can be computed in just 35 μs. For simplicity, assume that every iteration takes 35 μs—that's not strictly true, since it doesn't take as much time to do the first (k_1) as the last (k_{10}). However, by that standard, it would take only 350 μs to do all ten coefficients. So the actual time is really less than 350 μs. Now the ten filter coefficients are,

$$a_1 = a_1(10) = a_1(9) + k_{10}a_9(9)$$

$$a_2 = a_2(10) = a_2(9) + k_{10}a_8\ (8)$$

$$\vdots$$

$$a_9 = a_9(10) = a_9(9) + k_{10}a_1(9)$$

$$a_{10} = a_{10}(10) = -k_{10}.$$

encoding and other operations. All that remains of this analysis is to encode each 22.5-ms speech segment into the number of bits necessary to operate the system at 2400 bits/s.

Within 1 s, there are 44.5 time frames, each 22.5 ms long. At a rate of 2400 bits/s, each time frame holds 54 bits (2400/44.5), so the filter coefficients, gain, pitch and any other data must be encoded into 54 bits of information. Here's how the 54 bits are apportioned:

Filter coefficients a_1 through a_4	5 bits each
Filter coefficients a_5 through a_8	4 bits each
Filter coefficient a_9	3 bits
Filter coefficient a_{10}	2 bits
Gain	5 bits
Pitch and other data	8 bits
Total	54 bits

Now you've completed the examination of a speech compression system that transmits voice signals at 2400 bits/s rather than the 64 kbits/s previously required. What you need to know is what a practical speech compression system looks like.

Speech processing architecture

To send voice signals over transmission lines, you need a speech analysis circuit at the transmitting station and a speech synthesizer at the receiving end. (See Fig. 9 for a more detailed speech compression system than shown in Fig. 6.)

Both the analyzer and synthesizer circuits can be implemented in the same way—with a microprocessor, dedicated multiplier, and memory and microprogramming chips (as shown in the system block diagram of Fig. 10). The processor operates via a single data bus to take advantage of the multiplier's single-bus structure. And, as you can see, memory chips make up a large portion of the system.

Main memory, 2048 words × 16 bits, is a TMS 4048 chip (Texas Instruments). Supporting the main memory is a 16-word × 16-bit high-speed buffer memory (74LS219, also TI). And the microprogram memory, which accommodates up to 4096 words, is composed of eleven chips—each a l-k × 4 PROM (Monolithic Memories 63RS441). The processor itself, a 2910 microprogram controller (AMD), handles the 4096 words of the PROM. If necessary, you can expand the microprogram memory to include more bits/word, or to increase the number of control-instruction words. The system, as shown, operates in a 100-ns microcycle time.

This speech compressor is an example of what designing with microprocessors and multipliers devices can accomplish—a system design that employs a total IC count of under thirty chips. Previously, with only MSI devices available, you would have needed about 200 chips to do the same thing. ■■

Reprinted from Electronic Design-February 1, 1979